the Availability Digest

www.availabilitydigest.com

Defining Active/Active

Revision 1

January 2010

Last month, we started a discussion of the definition of active/active systems.[1] We noted that the general consensus is probably that it is a technique for building extremely reliable computing systems. But when we probed deeper, we started to find the caveats – the limitations imposed by one technology or another to achieve such high reliability. How far can these caveats reach before we must conclude that a particular approach is really not suitable for our applications because of the approach's limitations?

We took a first pass at an active/active definition and started a LinkedIn *Continuous Availability Forum* to elicit discussion on this topic. To join us, search on Continuous Availability Forum under "Groups" in LinkedIn.[2]

Several excellent and constructive suggestions were made, and we incorporate them into this Revision 1 of the document. As more suggestions come in, we will continue to keep the document updated.

A Reader's Dilemma

Rich Rosales of U.S. Bank, one of the Digest's readers, expressed this concern to us recently in some detail:[3] Excerpts from his comments follow (his full comments are contained in the original version of this document.[1]

> *The value of providing active/active has risen to the point of many software vendors claiming they support it - when in fact they do not. … In my recent experience, every vendor I spoke with claimed they supported an active/active architecture. When I pressed them for details on how they would handle collisions, they all asked "Why would you let that occur?" - as if the reality of updating the same record (near) simultaneously on two geographically distributed systems was simply unthinkable - "Wouldn't that be fraud?" one vendor asked.*
>
> *I think what we have here is an opportunity to help clarify the different levels of active/active. … We need to come up with new terminology - to prevent vendors from flooding the market*

[1] Defining Active/Active, *Availability Digest*; December 2009. http://www.availabilitydigest.com/public_articles/0412/defining_active_active.pdf

[2] Or go to http://www.linkedin.com/groupsDirectory?results=&sik=1260921605283&pplSearchOrigin=GLHD&keywords=continuous+availability+forum.

[3] See Rich's description of his experiences with implementing an active/active system, published in U.S. Bank Critiques Active/Active, *Availability Digest*; May 2009. http://www.availabilitydigest.com/public_articles/0405/usbank.pdf

with active/active … architectures, and provide a means by which those vendors that can and do provide higher levels of availability can distinguish themselves.

The Gold Standard – the "Perfect" Active/Active System

We agree wholeheartedly with Rich. Those of you who have been involved intimately with active/active systems understand the many shades of the term.

To get a handle on categorizing active/active systems, let us first define the "perfect" active/active system. If we can agree on such a reference definition, we can measure any architecture against that definition and can decide where the architecture complies and where it falls short. This will give us a level playing field when comparing active/active technologies.

We submit that a "perfect" active/active system will have the following attribute:

An active/active system comprises multiple geographically-distributed processing nodes using geographically-distributed consistent copies of the application database, such that the application network survives any single point of failure with no data loss. Furthermore, a transaction can be directed to any processing node in the application network; and the recovery time from the failure of a processing node or database copy is short enough that users are not aware of the fault.

A "perfect" active/active system meeting this attribute has many desirable characteristics:

(1) Disaster Tolerance - The system is disaster-tolerant since it comprises fully redundant, geographically-distributed components (processing nodes and database copies) and will survive the failure of any one component (or more if properly architected).

(2) No Data Loss - Should the system suffer a fault, no data is lost. All completed transactions or updates survive in all database copies.

(3) No Idle Nodes - All processing nodes and all database copies are available to equally share the application load since reads and updates can be routed to any processing node. The amount of excess capacity that must be licensed decreases with the number of nodes in the network since the loss of a node results in a smaller decrease in overall capacity.

(4) Locality – Database requests can be directed to the closest processing node to maximize performance. Processing nodes can use the closest database copy.

(5) Load Balancing -The system can be easily load balanced by redirecting read and update activity.

(6) Scalability - The system can be easily scaled since processing nodes and database copies can be added and the load rebalanced.

(7) No Unplanned Downtime - Unplanned downtime can be eliminated since transactions can be routed around a node failure with no impact on users.

(8) No Planned Downtime - Planned downtime can be eliminated by redirecting traffic prior to the scheduled outage, downing a node, upgrading it, and returning it to service while the other nodes handle the application workload.

We understand that in today's world, no technology achieves this goal.[4] However, different active/active technologies achieve different elements of this attribute; and users can select a technology that best suits their needs.

Categorizing Active/Active Architectures

The first thing to recognize is that there is a natural dichotomy in active/active architectures – asynchronous systems and synchronous systems.

Asynchronous Active/Active Systems

In an *asynchronous active/active system*, a change made to a source database copy is replicated to the target database copies after-the-fact. There are two primary methods for implementing an asynchronous active/active system – data replication and transaction replication.

Data Replication

Using data replication, changes (inserts, updates, deletes) made to one database are queued for transmission to the other database copies in the application network. A replication engine reads changes from the change queue of the source database and sends them to a target system, where they are applied to the target database.

There are several ways that changes can be organized for replication:

- Changes may be replicated independently one at a time.
- Changes may be replicated independently one at a time as they are made at the source system but are contained within the scope of a transaction. The begin and commit directives for the transaction are replicated independently. All changes within the scope of the transaction are committed (or aborted) at the target system when a commit (or abort) directive is replicated to the target system. (Note that the begin-transaction directive may by implied by the first change.)
- Changes within the scope of a transaction successfully committed at the source system may be sent as a batch of changes to be applied to the target database by the target system.
- Operations may be replicated. (Note: If these are commutative operations that can be applied in any order, such as add/subtract or multiply/divide, data collisions discussed later can be avoided).
- SQL operations may be replicated.

Transaction Replication

With transaction replication, an entire transaction is routed simultaneously to all nodes in the application network. Each node will execute the transaction independently.

[4] Though a technology called "coordinated commits," to be introduced perhaps in the next year, may substantially close this gap. See HP's NonStop Synchronous Gateway, *Availability Digest*; June 2009. http://www.availabilitydigest.com/public_articles/0406/hp_srg.pdf

Replication Latency

In either event, data replication or transaction replication, database changes are made at different times to different database copies. The difference in update times caused by asynchronous replication is called *replication latency.* Replication latency has the effect that nearly simultaneous updates to a data item by different processing nodes might be made in different order, leaving the database copies in different states. This is called a *data collision* and can lead to distributed database corruption.

Furthermore, if data replication is used to keep the databases in synchronism, there can be data loss if a node fails, as some of the data committed at the failed node may not have been replicated to the other nodes yet. This is not true of transaction replication if transactions are routed by the network or client and not by forwarding them from the receiving node.

Synchronous Active/Active Systems

In a *synchronous active/active system*, an update must be successfully applied to all database copies in the application network before the update is complete. Therefore, data collisions and data loss are eliminated.

However, synchronous replication creates another problem. The application is delayed as it waits for its changes to be applied across the application network, thus affecting performance. This delay is known as *application latency* and is strongly influenced by the distance separating the nodes. The internodal distance in a synchronous network is typically limited to a few kilometers by these performance considerations. Consequently, disaster tolerance may be compromised.[5]

Split-Brain Mode

A common problem with active/active systems occurs if connectivity between the systems is lost. In this case, changes made by one node cannot be replicated to the other node. If the system is asynchronous, and if both nodes are allowed to continue in operation, their databases will diverge over time and must be resynchronized when the network is restored. Data collisions will have to be detected and resolved. This is called *split-brain mode*.

If the system is synchronous, it must first convert to asynchronous mode or else no transaction will be able to complete – the system will be down. It is now operating in split-brain mode, as described above.

If split-brain is allowed, and if collisions can occur, collisions must be resolved when the nodes are reconnected. If split-brain mode is not acceptable, separated nodes must be shut down until connectivity is restored.

Let us look independently at asynchronous and synchronous architectures.

Asynchronous Active/Active Architectures

As we have discussed earlier, asynchronous active/active systems may be implemented in one of two ways. One way is to allow a transaction to be processed by any node and to replicate the updates created by the transaction to the other nodes via an asynchronous replication engine. We call this *data replication.* The other way is to send the entire transaction to all nodes, which will independently process the transaction. We call this *transaction replication*. Both have their advantages and disadvantages.

[5] New synchronous algorithms being developed may lessen this problem. See Footnote 3.

Data Replication

Asynchronous active/active architectures are generally implemented with asynchronous replication engines such as those from HP, IBM, Oracle, Gravic, Network Technologies, Oracle, Double-Take, and Continuent. In all cases, changes made to one database copy (which we will call the *source database*) are placed in a change queue of some sort and then sent after-the-fact over the replication network to the other (*target*) database copies in the application network.

The result is that all database copies are kept in synchronism in "near-real time." By near-real time, we mean that changes made to one database copy are reflected in the other database copies after a short delay - the replication latency. Replication latency includes not only the processing and intermediate storage delays in the replication engine but also the communication channel delay as changes propagate across the replication network (the *communication latency*).

Replication latency has two undesirable consequences:

(1) Should a processing node fail, all the changes that are currently in its replication pipeline will be lost.

(2) It is possible for two nodes to update the same data item at approximately the same time (within the replication latency interval). Each change will be replicated to the other system and will overwrite the original change there. As a consequence, both database copies are different; and both are wrong.

To handle the problem of data collisions, many asynchronous replication engines incorporate data collision and resolution facilities. Once detected, a collision is resolved by business rules either built into the replication engine, added via user exits, or performed manually.

Data loss and uncorrected data collisions result in a corrupted distributed database. Thus, there are two important aspects to understand with respect to an asynchronous data replication engine:

(1) What is its replication latency? This will determine the expected data loss following a node failure and the data collision rate.

(2) What are its capabilities to avoid or to identify and resolve data collisions?

The first aspect is simply a number (though it is system-dependent and may not be easily determined). It is the second aspect that leads to a variety of active/active architectures that must be clearly understood.

Transaction Replication

With transaction replication, there is no data loss if transactions are routed to the different nodes by the client or by the network. If instead, transactions are routed by the nodes, data loss may occur if the transaction is processed by a node before it is routed.

There is also a replication latency interval with transaction replication, but it is in the time interval during which different nodes process the transaction. The longer this interval, the more likely it is that transactions will be applied in a different order, causing data collisions to occur. However, with transaction replication, there is no way to detect a data collision. Therefore, data collisions cannot be corrected as they occur. Many systems employ a validation utility to compare the database copies so that discrepancies may be corrected after the fact.

Categorization of Asynchronous Active/Active Systems

We suggest the following categorizations of active/active architectures. The category names use the following nomenclature:

Nomenclature			Description
A			asynchronous replication
S			synchronous replication
	A		amorphous database (update anywhere) – a transaction can be routed to any node.
	P		partitioned database (update specific) – a transaction must be routed to the node owning the partition.
	T		transaction replication (update all) – a transaction must be routed to all nodes.
		+	data collision detection and resolution supported
		-	data collisions not supported

This leads to four asynchronous active/active architectures:

- AA+: Amorphous database with collision support.
- AA-: Amorphous database without collision support.
- AP-: Partitioned database without collision support.
- AT-: Transaction replication.

There is no AP+ architecture since the intent of partitioning is to avoid collisions.

There is no AT+ architecture since with transaction replication (as described above), there is no way to detect data collisions. Each application processes each transaction to completion and does not share or receive database information from the other application copies. In fact, one application copy might commit a transaction and another abort it; and neither will know about the actions of the other.

In the following descriptions of these architectures, the active/active attributes of the approach are described followed by the characteristics of the replication engine required to support the architecture.

<u>Architecture AA+: Amorphous Database with Collision Support</u>

Architecture

The database is fully distributed and appears amorphous to the application. Any transaction, update, or read can be applied to any database in the application network and is typically directed to the closest processing node. Data collisions are anticipated, and they are detected and resolved by facilities built into the replication engine.[6] Should a node fail, transactions are automatically routed to surviving nodes.

[6] Chapter 4, <u>Active/Active Topologies</u>, *Breaking the Availability Barrier II: Achieving Century Uptimes with Active/Active Systems*, AuthorHouse; 2007.

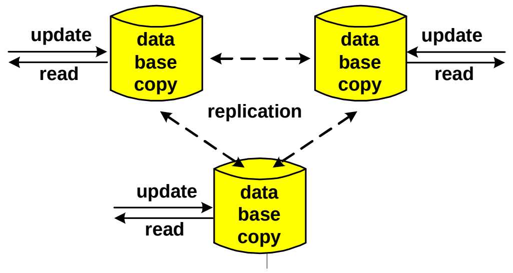
Replication Engine

This architecture requires bidirectional replication.[7] All database copies must be open for full read/write access by the applications. The architecture also requires that full data collision detection and resolution be supported by the replication engine.

Architecture AA-: Amorphous Database without Collision Support

Architecture

This architecture assumes that the application is such that collisions will not occur or that they may be ignored. Any transaction, update, or read can be applied to any database in the application network and is typically directed to the closest node. Data collisions are not anticipated, nor are they handled. Should a node fail, transactions are automatically routed to surviving nodes.



AA+ and AA- Architecture

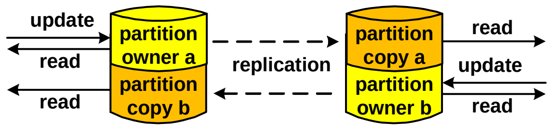
Replication Engine

This architecture requires bidirectional replication. All database copies must be open for full read/write access by the applications. Data collision detection and resolution need not be supported by the replication engine.

Architecture AP-: Partitioned Database without Collision Support

Architecture

The database is partitioned, and each partition is owned by one node. Only the owning node can update its partition, and all updates for a partition must be routed to the owning node. The owning node will replicate all updates to its partition to the other nodes. However, all nodes can read all partitions. Data collisions cannot occur. Provision must be made to detect the failure of a node. Should a node fail, its partition ownership must be transferred to another node.



AP- Architecture

Replication Engine

This architecture requires unidirectional replication. All database partitions must be open for read/write access on all nodes to support fast failover. Data collision detection and resolution are not required.

Architecture AT-: Transaction Replication without Collision Support

Architecture

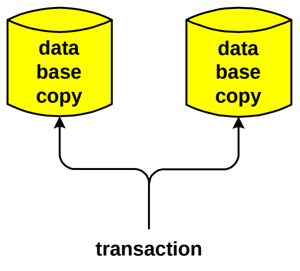
All transactions are sent simultaneously to every node in the application network by each client application. Each node executes the transactions or updates independently of the other

[7] Bidirectional replication implies that the replication engine prevents ping-ponging, or the return of a replicated data item to its source.

nodes. Updates submitted by different clients may be executed in different order by the nodes. Therefore, the databases may diverge over time due to data collisions unless the application is insensitive to transaction order.

Replication Engine

The replication engine required by this architecture submits transactions simultaneously to all nodes in the application network.



AT- Architecture

Master/Slave Architectures

There are other active/active architectures known as “master/slave.” In these systems, one node is the master. It controls all updates and replicates updates to the slave systems. There are two types of master/slave architectures:

- In one architecture, all nodes – master and slaves – can update their local databases. Slave updates are replicated to the master node, which resolves all data collisions. Its updates are replicated back to the slaves, including the originating slaves, to maintain a consistent database in the application network.[8] In this case, update activity can be balanced across all nodes as transactions can be routed to any node. These are AA+ systems differentiated by dual replication and by a single point of collision detection and resolution. The master database can be considered the database of record.

- In another architecture, all updates are sent directly to the master, which replicates its changes to its slaves. This is an AP- architecture with a single partition. It is often used in *update seldom, read often* applications to distribute read activity.

Asynchronous Summary

These levels are summarized in the following table, which shows the support for the desirable active/active characteristics.

Archi-tecture	Fast Failover	Disaster Tolerance	Data Loss Avoided	Data Collision Support	Balance Updates	Balance Reads	Scalable	Locality (route tx anywhere)	Split-Brain Avoided
AA+	Y	Y	N	Y	Y	Y	Y	Y	N
AA-	Y	Y	N	N	Y	Y	Y	Y	N
AP-	Y	Y	N	N	Y*	Y	Y*	N	Y
AT-	Y	Y	Y	N	N	Y	N	N	N**

* only by repartitioning the database.
** AT- is always in split-brain mode

Asynchronous Active/Active Characteristics

The architectures have different characteristics:

- AA-, AP-, and AT- cannot be used if data collisions are a concern or cannot be managed.
- AA+ and AA- allow a transaction to be routed to any node.

[8] HP’s OpenCall INS Goes Active/Active, *Availability Digest*, June 2007.
http://www.availabilitydigest.com/private/0206/motorola.pdf

- AT- requires that transactions be routed to all nodes.
- All levels support read load balancing.
- AA+ and AA- support update load balancing.
- AP- requires that the database be repartitioned to balance update load.
- AT- cannot be balanced for update load.
- AA+ and AA- databases will diverge if replication is lost (split-brain mode).
- AP- prevents split-brain database divergence.
- AT- is always in split-brain mode.

In general, an asynchronous replication engine can be categorized by the active/active architecture that it supports (its level) and its inherent replication latency (exclusive of communication latency). For instance, a particular replication engine might be a 100 millisecond AA+ asynchronous engine.

Note that active/active architecture AA+ has all of the characteristics of our definition of an ideal active/active system except for the potential of data loss following a node failure. An AA- architecture also meets this test if data collisions cannot happen or can be ignored.

Synchronous Active/Active Architectures

Though general-purpose synchronous replication engines can certainly be built, synchronous active/active architectures have followed a different path. Using today's available technology, most synchronous active/active systems are entire systems available from a system vendor. OpenVMS split-site clusters, IBM Parallel Sysplex, and Stratus Avance are examples of these.[9]

This may change when coordinated-commit synchronous replication engines become available.[4] These engines use asynchronous replication to propagate updates and to synchronize transactions only at commit time.

Recovery from a Node Failure

Synchronous replication faces a unique problem should a node fail or lose connectivity or should one node becomes so slow that it slows down all transactions. In this case, the system must remove that node from all further transactions and must switch to asynchronous replication to that node so that transaction processing can continue. This leads to split-brain mode; and if this is not acceptable, the disconnected or malfunctioning node must be shut down.

Recovery from a node failure is far more complex in synchronous recovery than in asynchronous recovery. During resynchronization, queued changes are replicated asynchronously from the operating nodes to the node being recovered. When the node is recovered, replication switches to synchronous mode.

This leads to a synchronous recovery problem. There are two approaches to the transition from asynchronous recovery to synchronous production:

(1) When the recovery change queues are sufficiently short, pause all operations until the resynchronization is complete. Then resume operations in synchronous mode.

[9] OpenVMS Active/Active Split-Site Clusters, *Availability Digest*; June 2008.
http://www.availabilitydigest.com/public_articles/0306/openvms.pdf
Parallel Sysplex – Fault Tolerance from IBM, *Availability Digest*; April 2008.
http://www.availabilitydigest.com/public_articles/0304/ibm_sysplex.pdf
Stratus' Avance Brings Availability to the Edge, *Availability Digest*; February 2009.
http://www.availabilitydigest.com/public_articles/0402/avance.pdf

(2) If there are long transactions (such as batch transactions), pausing the system prior to switching to synchronous replication may not be feasible. In this case, operations must continue while the recovering system is receiving both asynchronous and synchronous replicated transactions. This may cause data collisions between asynchronous changes and synchronous changes.[10]

Categorization of Synchronous Active/Active Systems

Therefore, there is a form of data collisions in synchronous replication during recovery. This leads to two categories of synchronous replication engines:

S+: Operations are not paused during recovery. Asynchronous/synchronous data collisions must be avoided or resolved.

S-: Operations are paused during recovery. There are no asynchronous/synchronous data collisions.

Synchronous Summary

Each synchronous technique varies in how it maintains database copies in synchronism and how it resynchronizes following a node or network fault. However, the result is the same. The active/active attribute is met in full except for the limits on disaster tolerance. There is no data loss. There are no data collisions. Failover is fast, and system upgrades can be rolled through the application network one node at a time to eliminate planned downtime. Read and update activity can be uniformly distributed across the nodes, and the system is scalable by adding nodes.

Archi-tecture	**Fast Failover**	**Disaster Tolerance**	**Data Loss Avoided**	**Data Collision Support**	**Balance Updates**	**Balance Reads**	**Scalable**	**Locality (route tx anywhere)**	**Split-Brain Avoided**
S+	Y	Y	Y	Y	Y	Y	Y	Y	N
S-	Y	Y	Y	Y	Y	Y	Y	Y	N

Synchronous active/active architectures generally meet all of the characteristics of our ideal active/active architecture except for limits on geographical separation. Since a transaction or an update must be completed on all copies of the database across the application network, transaction response times are increased. Most vendors of synchronous replication systems limit their support of geographical dispersion to some specified distance, which may not meet the disaster-tolerance requirements of the application.

Therefore, a convenient means of classification of synchronous replication systems is simply to state the distance limitation. For instance, a system may be a 20-kilometer S+ synchronous replication system.

Summary

This categorization of active/active technologies is still a work-in-progress. To those looking to implement active/active systems, a categorization that achieves substantial consensus is important to be able to weed out vendor offerings (or in-house implementation strategies) that will not meet the application's needs.

Please contribute to this important subject by sharing your comments with us on our LinkedIn Continuous Availability Forum at

[10] B. Highleyman, J. Hoffmann, P. Holenstein, Recovering from Synchronous Replication Failures, *The Connection*; September/October 2009.

http://www.linkedin.com/groupsDirectory?results=&sik=1260921605283&pplSearchOrigin=GLHD&keywords=continuous+availability+forum.
If this link doesn't work for you, search on Continuous Availability Forum under "Groups" in LinkedIn.